REMARKS

Reconsideration of the application is respectfully requested in view of the foregoing amendments and following remarks.

Claims 1-30 are currently pending in this application. Claims 1, 7, 13, 20, and 23 are independent. Claims 1-30 were rejected in the Office Action mailed November 3, 2005 ["the Action"], in view of different combinations of references. The Applicants respectfully disagree.

I. Amendments

No claims have been amended. No new matter has been added.

II. Cited Art

The Applicants make the following observations in the interest of reaching a shared understanding of the application as well as of the disclosures of U.S. Patent No. 5,467,472 to Williams et al., U.S. Patent No. 6,701,352 to Gardner et al., and U.S. Patent No. 6,389,491 to Jacobson et al.

A. U.S. Patent No. 5,467,472 to Williams et al. ["Williams"]

Williams describes various features and benefits of a system for "generating and maintaining property sets." [Williams, 4:33-49.] Williams discussion of property set generation and maintenance does not mention either compilation or generation of interface implementations.

Williams only mention of compilation or interfaces comes in the course of providing background information. There, Williams generally describes "an overview of well-known object-orientated programming techniques." [Williams, 1:46-47.] In the course of this overview, Williams mentions U.S. Patent No. 5,297,284, entitled "A Method for Implementing Virtual Functions and Virtual Bases in a Compiler for an Object-Oriented Programming Language," as describing a particular object layout. [Williams, 2:39-44.]

Williams then continues describing background information, including the general use of interfaces to share objects and that interfaces may be used to provide object access at compile time. [Williams, 2:47-67.] Thus, Williams describes what interfaces are, generally, as well as that objects can be instantiated from classes which implement defined interfaces:

> *To allow an object of an arbitrary class to be shared with a client program, interfaces are defined through which an object can be accessed without the need for the client program to have access to the class definitions at compile time. An interface is a named set of logically related function members....* Thus, an interface provides a published protocol for two programs to communicate. Interfaces are typically used for derivation: a program defines (implements) classes that provide implementations for the interfaces the classes are derived from. Thereafter, objects are created as instances of these derived classes. Objects instantiated from a derived class implementing particular interfaces are said to "support" the interfaces.

[Williams, 2:47-66.] Williams also describes interfaces being used to describe function members without dependence on the members' implementations, through information which is kept in a persistent registry.

> When a client program desires to share an object, the client program needs access to the code that implements the interfaces for the object (the derived class code). *To access the derived class code (also referred to as class code), each class implementation is given a unique class identifier (a "CLSID").* For example, code implementing a spreadsheet object developed by Microsoft Corporation may have a class identifier of "MSSpreadsheet," while code implementing a spreadsheet object developed by another corporation may have a class identifier of "LTSSpreadsheet." *A persistent registry in each computer system is maintained that maps each CLSID to the code that implements the class. Typically, when a spreadsheet program is installed on a computer system, the persistent registry is updated to reflect the availability of that class of spreadsheet objects.* So long as a spreadsheet developer implements each function member defined by the interfaces to be supported by spreadsheet objects and so long as the persistent registry is maintained, the client program can access the function members of shared spreadsheet objects without regard to which server program has implemented them or how they have been implemented.

[Williams, 3:1-22.]

Thus, according to Williams, the classes that implement interfaces are known before runtime and identifiers of the classes are kept in a registry in order to make that information readily available at runtime. This section of Williams does not relate to generating interface implementation code. After the background, Williams does not again describe interfaces, except to mention specific instances of interfaces. Williams also does not further discuss compilation and generation of interfaces.

B. U.S. Patent No. 6,701,352 to Gardner et al. ["Gardner"]

Gardner describes various aspects of an Object Linking and Embedding Automation facility in Microsoft Windows 95. It describes this as an example of "bridge software" which "enables one application program to communicate with another application program through an agreed-upon, shared inter-application communication scheme." [Gardner, 7:46-50.] It further describes bridge software during *runtime operation* of applications:

> In one embodiment, the bridge software 210 provides a distributed object communication facility. An application program can write data to an object, and invoke a transport routine to cause the bridge software 210 to transport the object to another application program. The receiving application program extracts a message and data from the object and acts upon them.

[Gardner, 7:50-56.]

With regard to OLE in particular, Gardner describes the use of "dispatchable interfaces" and "late binding":

> Using dynamic dispatching under OLE, an automation client can invoke a method or manipulate a property of a server component by a late binding mechanism. *At run time, the automation client obtains a dispatch identifier from a type library* associated with the server component. The dispatch identifier is passed to an "invoke" method of OLE Automation that resolves which method of the server component to call at run time. *The type library is created at run time* by an object definition language (ODL) file that describes interfaces of the server component.

[Gardner, 8:4-14, emphasis added.]

While cited passages of Gardner discuss runtime use of dispatchable interfaces and late binding (e.g., using the Invoke method), they do not discuss generation of interface implementations. What they specifically *do not* describe is any method of generating dispatch interface implementations, whether from definition information or otherwise. It is also worth noting that Gardner does not describe specifics of any particular interfaces, but rather methods of late binding to the particular interfaces using type libraries at run time, where the late binding methods are described by definition language.

C. U.S. Patent No. 6,389,491 to Jacobson et al. ["Jacobson"]

Jacobson describes the use of a dual interface mechanism:

> As known by those skilled in the art, ActiveX Automation Servers by definition support dual interfaces, including custom vtable-based access interfaces such as the IIO interface 38b and the well-known IDispatch interface which supports method invocation for applications written in languages that do not support vtable access. Thus, implementing the universal I/O interface 30 as an ActiveX Universal I/O Automation Server allows any language that can support vtable access to access the methods directly via the vtable. Languages that support late binding can invoke the methods via the Invoke( ) method of the IDispatch interface.

[Jacobson, 4:65-5:9.]

Thus, Jacobson describes both direct access to interface methods using the vtable and late-bound access using IDispatch interface.

III. Features of the Instant Application

For the sake of illustration and without implying limitations on the claims of the instant application, Applicants provide the following comments. The instant application describes automatic generation of late binding interface implementations. [Application, page 13, lines 3-11.] In one example, the application describes automatic generation of a dispatch interface implementation by a C++ compiler based on definition information for a dispatch interface and C++ code. [Application, page 9, line 22 to page 10, line 2.] As the application describes, this definition information can be embedded in C++ code. [Application, page 16, lines 10-23; Figure 5.] This automatic generation provides benefits over other mechanisms for providing late binding interfaces. For example, manual programming of late binding interfaces has its difficulties:

> Manually programming the extra layers of code for late binding interfaces has proven to be notoriously difficult and time-consuming for programmers. Writing the corresponding client side code for packing arguments to and unpacking arguments from the generic data structures has also proven difficult.

[Application, page 5, lines 18-21.] As the Application also notes, other mechanisms which seek to provide non-manual support for implementation of late binding interfaces also suffer from disadvantages. For instance, the use of type libraries can result in excessive run-time calls during execution, which can slow performance. [Application, page 6, lines 9-12.] As another example, wizard-created dispatch maps can be slow, and can result in code which is confusing and unstable if later modified by a programmer. [Application, page 7, lines 4-12.]

The Application describes examples of a compiler which overcomes these disadvantages by using definition information to generate "potentially thousands of lines of code that a programmer previously had to write when creating a dispatch interface implementation." [Application, page 15, line 23 to page 16, line 2.] This is done automatically, in one implementation, by modules in a compiler which recognize definition information in C++ source code, and process the C++ code and definition information to produce a late binding interface. [Application, page 19, lines 1-17.] This technique provides a simple, systematic process which additionally prevents errors due to incomplete implementation. [Application, page 8, lines 7-10.]

As the application describes, in one example the modules perform these processes through automatic parsing of definition information and source code:

> A front end module 722 reads and performs lexical analysis upon the file 710. Basically, the front end module 722 reads and translates a sequence of characters in the file 710 into syntactic elements, or "tokens," indicating constants, identifiers, operator symbols, keywords, punctuation, etc.
>
> A converter module 724 parses the tokens into an intermediate representation. For tokens from C++ source code, the converter module 724 checks syntax and groups tokens into expressions or other syntactic structures, which in turn coalesce into statement trees. Conceptually, these tress form a parse tree 732.

[Application, page 20, lines 4-12.] Another example of parsing is demonstrated in Figure 8, where IDL attribute information is parsed at act 850. [Application, Figure 8, page 24, lines 14-15.]

The products of this parsing are then used to create an output dispatch interface implementation 790: "Based upon the symbol table 730 and the parse tree 732, a back end module 726 translates the intermediate representation of file 710 into output code." [Application page 21, lines 1-3.] Thus, a late binding interface implementation is generated automatically, providing the benefits described above.

IV. Claims 1-4, 6, and 26

Claim 1 is directed to generating a dispatch interface implementation. Definition information is received which defines dispatch interface features of a dispatch interface that includes dispatch methods. A dispatch interface implementation for operating one or more other methods is generated, where the generating includes parsing the definition information as well as

the programming language code (for the one or more other methods) during compilation. The implementation includes executable code for: (1) the one or more other methods, (2) a dispatch method for mapping names (of the one or more other methods) to dispatch identifiers for binding at run time, and (3) a dispatch method for calling the one or more other methods at run time responsive to client requests.

The Action rejects claims 1-4, 6, and 26 as being unpatentable over Williams in view of Gardner. The Applicants respectfully disagree. For at least the following reasons, claim 1 should be allowable.

A. Williams and Gardner, taken separately or in combination, fail to teach or suggest at least one limitation of claim 1.

Claim 1 recites at least one limitation that Williams and Gardner, taken separately or in combination, fail to teach or suggest. For example, claim 1 recites "generating a dispatch interface implementation for operating the one or more methods, *wherein the generating includes parsing the programming language code and the definition information during compilation.*" No portion of Williams was cited against the emphasized language by the Examiner, and the Action indicates "Williams did not explicitly state dispatch interface from the definition information." [Action, page 3.] And even if, for the sake of argument, Williams disclosed interface definitions for function members of an interface, this does not teach or suggest "definition information that defines dispatch interface features" as recited in claim 1.

Gardner, in turn, does not relate to generation of a dispatch interface implementation or any activities during compilation, including parsing of programming language code or definition information. Gardner is even further from teaching or suggesting generation of a dispatch interface during compilation, where that generation includes parsing definition information and programming language code, as recited in claim 1. In fact, Gardner's discussion of obtaining dispatch identifiers from *type libraries created at run-time* leads directly away from the above-cited "compilation" language of claim 1. [*Compare* the "type library implementation" technology described in the backgrounds of the present application at page 6, lines 5-12.]

Because Williams and Gardner taken separately fail to teach or suggest the above-cited language of claim 1, the combination of Williams and Gardner also fails to teach or suggest the above-cited language of claim 1, and claim 1 should be allowable.

B. The combination of Williams and Gardner is improper.

The combination proposed by the Examiner to reject claim 1 is improper. The Examiner admits that Williams does not disclose "dispatch interface from the definition information." [Action, page 3.] The Applicants agree. The Examiner argues, however, that Gardner demonstrates "dispatch interface" features and that:

> [i]t would have been obvious to one of ordinary skill in the art at the time of the invention to implement the abstract class system of Williams with a description of dispatch late binding interface to implement as found in Gardner's teaching. This implementation would have been obvious because one of ordinary skill in the art would be motivated to provide object interface for as many environments as possible in order to facilitate greater acceptance and use in the marketplace.

[Action, pages 3-4.]

Even if, for the sake of argument, Williams *could be* modified as suggested by the Examiner, this is not enough to make the Examiner's proposed modification obvious. [MPEP 2143.01; *see also* MPEP 2142.01 and 2145.X.C and D.] In fact, to the extent that Wiliams does discuss the use of interfaces, the Examiner's proposed modification *changes the principle of operation* of Williams and is thus improper. [*See In re Ratti*, 270 F.2d 810, MPEP § 2143.01.] In addition, Williams and Gardner teach away from the combination suggested by the Examiner.

Williams is directed toward generating and maintaining property sets. It is not directed toward compilation techniques or structures, and only briefly discusses compilation as a general background matter. Additionally, Williams contains only brief descriptions of the use of interfaces, as well as references to specific interfaces. Gardner, by contrast, describes the use of bridge software during runtime. There is no motivation in either Williams or Gardner to create a combination.

Furthermore, Williams describes interfaces whose function members are known at compile time, and for which persistent information is kept in order to facilitate runtime execution. [Williams, 3:1-22.] The Examiner's proposed modification would change this principle of operation of Williams. The Examiner proposes modifying Williams according to Gardner, so as to allow a client and server to communicate by using dispatchable interfaces for which interface information is *obtained at run time from a type library that is not created until*

run time. [Gardner 8:7-14.] According to late binding as in Gardner the registry information of Williams would be unnecessary.

Williams use of persistent registry data to maintain interface information suggests, if nothing else, knowing interface function members before execution. Because of this, Williams teaches away from creation of a type library at run time whose purpose is to provide interface information, as this would be duplicative and inefficient. Thus, Williams teaches away from combination with Gardner, which uses these runtime techniques.

Finally, inasmuch as Gardner describes a type library implementation for a dispatch interface, it also leads away from the modification the Examiner has made. Creating a type library at run time leads away from modifying a compiler to generate dispatch interface implementations.

For at least these reasons, claim 1 should be allowable.

Claims 2-4, 6, and 26 (which depend from claim 1) should also be allowable, but the Applicants will not belabor the merits of the separate patentability of claims 2-4, 6, and 26.

V. Claim 5

The Action rejects claim 5 (which depends from claim 1) as being unpatentable over Williams in view of Gardner and Jacobson. The Applicants respectfully disagree.

Taken separately or in combination with the other references, Jacobson does not teach or suggest the above-cited language of claim 1 that Williams and Gardner fail to teach or suggest (see section IV). Moreover, the combination of Williams with Gardner and Jacobson is improper for at least the reasons that the combination of Williams with Gardner is improper (see section IV).

The Applicants will not belabor the merits of the separate patentability of claim 5. Claim 5 should be allowable.

VI. Claims 7-11 and 27

Claim 7 is directed to a compiler system that generates a late binding interface implementation. The compiler system includes a front end module, a converter module, and a back end module. The front end module receives definition information (defining late binding interface features of a late binding interface) and programming language code (for implementing

one or more late bound methods). The converter module identifies relations between the definition information and the one or more late bound methods during compilation, including parsing the definition information and the programming language code. The back end module generates a late binding interface implementation based upon the relations, for operating the one or more late bound methods.

The Action rejects claims 7-11 and 27 as being unpatentable over Williams in view of Gardner. The Applicants respectfully disagree. For at least the following reasons, claim 7 should be allowable.

A. Williams and Gardner, taken separately or in combination, fail to teach or suggest at least one limitation of claim 7.

Claim 7 recites at least one limitation that Williams and Gardner, taken separately or in combination, fail to teach or suggest. For example, claim 7 recites a "compiler system" that includes a "converter module that identifies relations between the definition information and the one or more late bound methods during compilation, including parsing the definition information and the programming language code" and "a back end module that generates a late binding interface implementation based upon the relations." The Action indicates "Williams did not explicitly state late bound." [Action, page 6.] And even if, for the sake of argument, Williams disclosed interface definitions for function members of an interface, this does not teach or suggest "definition information defining late binding interface features" as recited in claim 7.

Gardner, in turn, does not relate to compiler systems or parsing of programming language code or definition information. Gardner is even further from teaching or suggesting a compiler system that parses definition information as recited in claim 7. In fact, Gardner's discussion of obtaining dispatch identifiers from *type libraries created at run-time* leads directly away from the above-cited "compiler system" language of claim 7. [*Compare* the "type library implementation" technology described in the backgrounds of the present application at page 6, lines 5-12.]

Because Williams and Gardner taken separately fail to teach or suggest the above-cited language of claim 7, the combination of Williams and Gardner also fails to teach or suggest the above-cited language of claim 7, and claim 7 should be allowable.

B. The combination of Williams and Gardner is improper.

The combination proposed by the Examiner to reject claim 7 is improper. The Examiner admits that Williams does not disclose "late bound." [Action, page 6.] The Applicants agree. The Examiner argues, however, that Gardner demonstrates "dynamic/late binding, dispatchable" features and:

> [i]t would have been obvious to one of ordinary skill in the art at the time of the invention to implement the abstract class system of Williams with a description of dispatch late binding interface to implement as found in Gardner's teaching. This implementation would have been obvious because one of ordinary skill in the art would be motivated to provide object interface for as many environments as possible in order to facilitate greater acceptance and use in the marketplace. [Action, page 6.]

Even if, for the sake of argument, Williams *could be* modified as suggested by the Examiner, this is not enough to make the Examiner's proposed modification obvious. [MPEP 2143.01; *see also* MPEP 2142.01 and 2145.X.C and D.] In fact, to the extent that Wiliams does discuss the use of interfaces, the Examiner's proposed modification *changes the principle of operation* of Williams and is thus improper. [*See In re Ratti*, 270 F.2d 810, MPEP § 2143.01.] In addition, Williams and Gardner teach away from the combination suggested by the Examiner.

Williams is directed toward generating and maintaining property sets. It is not directed toward compilation techniques or structures, and only briefly discusses compilation as a general background matter. Additionally, Williams contains only brief descriptions of the use of interfaces, as well as references to specific interfaces. Gardner, by contrast, describes the use of bridge software during runtime. There is no motivation in either Williams or Gardner to create a combination.

Furthermore, Williams describes interfaces whose function members are known at compile time, and for which persistent information is kept in order to facilitate runtime execution. [Williams, 3:1-22.] The Examiner's proposed modification would change this principle of operation of Williams. The Examiner proposes modifying Williams according to Gardner, so as to allow a client and server to communicate by using dispatchable interfaces for which interface information is *obtained at run time from a type library that is not created until run time*. [Gardner 8:7-14.] According to late binding as in Gardner the registry information of Williams would be unnecessary.

Williams use of persistent registry data to maintain interface information suggests, if nothing else, knowing interface function members before execution. Because of this, Williams teaches away from creation of a type library at run time whose purpose is to provide interface information, as this would be duplicative and inefficient. Thus, Williams teaches away from combination with Gardner, which uses these runtime techniques.

Finally, inasmuch as Gardner describes a type library implementation for a dispatch interface, it also leads away from the modification the Examiner has made. Creating a type library at run time leads away from modifying a compiler to generate dispatch interface implementations.

For at least these reasons, claim 7 should be allowable.

Claims 8-11 and 27 (which depend from claim 7) should also be allowable, but the Applicants will not belabor the merits of the separate patentability of claims 8-11 and 27.

VII. Claim 12

The Action rejects claim 12 (which depends from claim 7) as being unpatentable over Williams in view of Gardner and Jacobson. The Applicants respectfully disagree.

Taken separately or in combination with the other references, Jacobson does not teach or suggest the above-cited language of claim 7 that Williams and Gardner fail to teach or suggest (see section VI). Moreover, the combination of Williams with Gardner and Jacobson is improper for at least the reasons that the combination of Williams with Gardner is improper (see section VI).

The Applicants will not belabor the merits of the separate patentability of claim 12. Claim 12 should be allowable.

VIII. Claims 13-18 and 28

Claim 13 is directed to generating a late binding interface implementation. Definition information defines late binding interface features of a late binding interface. A late binding interface implementation for operating one or more late bound methods is generated during compilation, where the generating includes parsing programming language code (for the one or more late bound methods) and definition information. The implementation includes one or more

late binding methods, including a method for calling the one or more late bound methods responsive to client requests.

The Action rejects claims 13-18 and 28 as being unpatentable over Williams in view of Gardner. The Applicants respectfully disagree. For at least the following reasons, claim 13 should be allowable.

For at least the following reasons, claim 13 should be allowable.

A. Williams and Gardner, taken separately or in combination, fail to teach or suggest at least one limitation of claim 13.

Claim 13 recites at least one limitation that Williams and Gardner, taken separately or in combination, fail to teach or suggest. For example, claim 13 recites "generating a late binding interface implementation for operating the one or more late bound methods, wherein the generating includes parsing the programming language code and the definition information during compilation." The Action indicates "Williams did not explicitly state late bound." [Action, page 8.] And even if, for the sake of argument, Williams disclosed interface definitions for function members of an interface, this does not teach or suggest "definition information that defines late binding interface features" as recited in claim 13.

Gardner, in turn, does not relate to generation of a late binding interface implementation with a compiler system, where that generation includes parsing of programming language code or definition information. Gardner is even further from teaching or suggesting a compiler system that parses definition information as recited in claim 13. In fact, Gardner's discussion of obtaining dispatch identifiers from *type libraries created at run-time* leads directly away from the above-cited "compilation" language of claim 13. [*Compare* the "type library implementation" technology described in the backgrounds of the present application at page 6, lines 5-12.]

Because Williams and Gardner taken separately fail to teach or suggest the above-cited language of claim 13, the combination of Williams and Gardner also fails to teach or suggest the above-cited language of claim 13, and claim 13 should be allowable.

B. The combination of Williams and Gardner is improper.

The combination proposed by the Examiner to reject claim 13 is improper. The Examiner admits that Williams does not disclose various "late bound" language in claim 13. [Action, page 7.] The Applicants agree. The Examiner argues, however, that Gardner demonstrates "dynamic/late binding, dispatchable" features and that:

> [i]t would have been obvious to one of ordinary skill in the art at the time of the invention to implement the abstract class system of Williams with a description of dispatch late binding interface as found in Gardner's teaching. This implementation would have been obvious because one of ordinary skill in the art would be motivated to provide object interface for as many environments as possible in order to facilitate greater acceptance and use in the marketplace." [Action, pages 7-8.]

Even if, for the sake of argument, Williams *could be* modified as suggested by the Examiner, this is not enough to make the Examiner's proposed modification obvious. [MPEP 2143.01; *see also* MPEP 2142.01 and 2145.X.C and D.] In fact, to the extent that Wiliams does discuss the use of interfaces, the Examiner's proposed modification *changes the principle of operation* of Williams and is thus improper. [*See In re Ratti*, 270 F.2d 810, MPEP § 2143.01.] In addition, Williams and Gardner teach away from the combination suggested by the Examiner.

Williams is directed toward generating and maintaining property sets. It is not directed toward compilation techniques or structures, and only briefly discusses compilation as a general background matter. Additionally, Williams contains only brief descriptions of the use of interfaces, as well as references to specific interfaces. Gardner, by contrast, describes the use of bridge software during runtime. There is no motivation in either Williams or Gardner to create a combination.

Furthermore, Williams describes interfaces whose function members are known at compile time, and for which persistent information is kept in order to facilitate runtime execution. [Williams, 3:1-22.] The Examiner's proposed modification would change this principle of operation of Williams. The Examiner proposes modifying Williams according to Gardner, so as to allow a client and server to communicate by using dispatchable interfaces for which interface information is *obtained at run time from a type library that is not created until run time*. [Gardner 8:7-14.] According to late binding as in Gardner the registry information of Williams would be unnecessary.

Williams use of persistent registry data to maintain interface information suggests, if nothing else, knowing interface function members before execution. Because of this, Williams teaches away from creation of a type library at run time whose purpose is to provide interface information, as this would be duplicative and inefficient. Thus, Williams teaches away from combination with Gardner, which uses these runtime techniques.

Finally, inasmuch as Gardner describes a type library implementation for a dispatch interface, it also leads away from the modification the Examiner has made. Creating a type library at run time leads away from modifying a compiler to generate dispatch interface implementations.

For at least these reasons, claim 13 should be allowable.

Claims 14-18 and 28 (which depend from claim 13) should also be allowable, but the Applicants will not belabor the merits of the separate patentability of claims 14-18 and 28.

IX. Claim 19

The Action rejects claim 19 (which depends from claim 13) as being unpatentable over Williams in view of Gardner and Jacobson. The Applicants respectfully disagree.

Taken separately or in combination with the other references, Jacobson does not teach or suggest the above-cited language of claim 13 that Williams and Gardner fail to teach or suggest (see section VIII). Moreover, the combination of Williams with Gardner and Jacobson is improper for at least the reasons that the combination of Williams with Gardner is improper (see section VIII).

The Applicants will not belabor the merits of the separate patentability of claim 19. Claim 19 should be allowable.

X. Claims 20-22 and 29

Claim 20 is directed to automatically generating an interface implementation having early binding and late binding mechanisms. Definition information defines late binding interface features of the interface. An interface implementation is generated at compile time for alternatively operating one or more methods by an early binding mechanism or by a late binding mechanism, where the generating includes parsing the definition information and programming language code (for the one or more methods of the interface). The early binding mechanism

provides for direct invocation of the one or more methods. (See, e.g., page 3 of the application, which describes the use of "virtual function tables" in early binding of method names to method code in an interface.) The late binding mechanism provides for invocation of the one or more methods responsive to a request through a late binding method.

The Action rejects claims 20-22 and 29 as being unpatentable over Williams in view of Gardner and Jacobson. The Applicants respectfully disagree. For at least the following reasons, claim 20 should be allowable.

A. Williams, Gardner, and Jacobson, taken separately or in combination, fail to teach or suggest at least one limitation of claim 20.

Claim 20 recites at least one limitation that Williams, Gardner, and Jacobson, taken separately or in combination, fail to teach or suggest. For example, claim 20 recites "generating an interface implementation for alternatively operating the one or more methods by an early binding mechanism or by a late binding mechanism, wherein the generating includes parsing the programming language code and the definition information during compilation." The Action indicates "Williams did not explicitly state late bound." [Action, page 16.] And, even if, for the sake of argument, Williams disclosed interface definitions for function members of an interface, this does not teach or suggest "definition information that defines late binding interface features" as recited in claim 20.

Gardner, in turn, does not relate to generation of an interface implementation or any compile time activities, or parsing of programming language code or definition information. Gardner is even further from teaching or suggesting the above-cited language of claim 20. In fact, Gardner's discussion of obtaining dispatch identifiers from *type libraries created at run-time* leads directly away from the above-cited "compilation" language of claim 20. [*Compare* the "type library implementation" technology described in the backgrounds of the present application at page 6, lines 5-12.] Likewise, Jacobson does not relate to generation of an interface implementation or compile time activities. Jacobson is even further from teaching or suggesting the above-cited language of claim 20.

Because Williams, Gardner, and Jacobson taken separately fail to teach or suggest the above-cited language of claim 20, the combination of Williams, Gardner, and Jacobson also fails to teach or suggest the above-cited language of claim 20, and claim 20 should be allowable.

B. The combination of Williams, Gardner, and Jacobson is improper.

The combination proposed by the Examiner to reject claim 20 is improper. The Examiner admits that Williams does not disclose various "late bound" language in claim 20. [Action, page 16.] The Applicants agree. The Examiner argues, however, that Gardner demonstrates "dynamic/late binding, dispatchable" features and:

> It would have been obvious to one of ordinary skill in the art at the time of the invention to implement the abstract class system of Wiliams with a description of dispatch late binding interface to implement as found in Gardner's teaching. This implementation would have been obvious because one of ordinary skill in the art would be motivated to provide object interface for as many environments as possible in order to facilitate greater acceptance and use in the marketplace. [Action, page 16.]

Even if, for the sake of argument, Williams *could be* modified as suggested by the Examiner, this is not enough to make the Examiner's proposed modification obvious. [MPEP 2143.01; *see also* MPEP 2142.01 and 2145.X.C and D.] In fact, to the extent that Wiliams does discuss the use of interfaces, the Examiner's proposed modification *changes the principle of operation* of Williams and is thus improper. [*See In re Ratti*, 270 F.2d 810, MPEP § 2143.01.] In addition, Williams and Gardner teach away from the combination suggested by the Examiner.

Williams is directed toward generating and maintaining property sets. It is not directed toward compilation techniques or structures, and only briefly discusses compilation as a general background matter. Additionally, Williams contains only brief descriptions of the use of interfaces, as well as references to specific interfaces. Gardner, by contrast, describes the use of bridge software during runtime. There is no motivation in either Williams or Gardner to create a combination.

Furthermore, Williams describes interfaces whose function members are known at compile time, and for which persistent information is kept in order to facilitate runtime execution. [Williams, 3:1-22.] The Examiner's proposed modification would change this principle of operation of Williams. The Examiner proposes modifying Williams according to Gardner, so as to allow a client and server to communicate by using dispatchable interfaces for which interface information is *obtained at run time from a type library that is not created until run time*. [Gardner 8:7-14.] According to late binding as in Gardner the registry information of Williams would be unnecessary.

Williams use of persistent registry data to maintain interface information suggests, if nothing else, knowing interface function members before execution. Because of this, Williams teaches away from creation of a type library at run time whose purpose is to provide interface information, as this would be duplicative and inefficient. Thus, Williams teaches away from combination with Gardner, which uses these runtime techniques.

Finally, inasmuch as Gardner describes a type library implementation for a dispatch interface, it also leads away from the modification the Examiner has made. Creating a type library at run time leads away from modifying a compiler to generate dispatch interface implementations.

The combination of Williams with Gardner and Jacobson is improper for at least the reasons that the combination of Williams with Gardner is improper.

For at least these reasons, claim 20 should be allowable.

Claims 21-22 and 29 (which depend from claim 20) should also be allowable, but the Applicants will not belabor the merits of the separate patentability of claims 21-22 and 29.

XI. Claims 23-25 and 30

Claim 23 is directed to automatically generating call site code for calling a late bound method through a late binding interface. Received definition information and programming language code (for calling a late bound method) are parsed. Based upon type information for one or more input arguments of the late bound method, code is generated for packing the one or more input arguments into a generic argument data structure. Code is also generated for calling the late bound method through an invocation method of the late binding interface, wherein the calling includes passing the generic argument data structure to the invocation method.

The Action rejects claims 23-25 and 30 as being unpatentable over Williams in view of Gardner. The Applicants respectfully disagree. For at least the following reasons, claim 23 should be allowable.

A. Williams and Gardner, taken separately or in combination, fail to teach or suggest at least one limitation of claim 23.

Claim 23 recites at least one limitation that Williams and Gardner, taken separately or in combination, fail to teach or suggest. For example, claim 23 recites "parsing the definition

information and the programming language code during compilation." The Action indicates "Williams did not explicitly state type information and late bound." [Action, page 11.] And, even if, for the sake of argument, Williams disclosed interface definitions for function members of an interface, this does not teach or suggest "definition information for late binding interface features" as recited in claim 23.

Gardner, in turn, does not relate to parsing of programming language code or definition information. Gardner is even further from teaching or suggesting parsing programming language code and definition information during compilation, as recited in claim 23. In fact, Gardner's discussion of obtaining dispatch identifiers from *type libraries created at run-time* leads directly away from the above-cited "compilation" language of claim 1. [*Compare* the "type library implementation" technology described in the backgrounds of the present application at page 6, lines 5-12.]

Because Williams and Gardner taken separately fail to teach or suggest the above-cited language of claim 23, the combination of Williams and Gardner also fails to teach or suggest the above-cited language of claim 23, and claim 23 should be allowable.

B. The combination of Williams and Gardner is improper.

The combination proposed by the Examiner to reject claim 23 is improper. The Examiner admits that Williams does not disclose a number of features, including:

> state type information and late bound and code for packing the one or more arguments into a generic argument data structure; and generating code for calling the late bound method through an invocation method of the late binding interface, wherein the calling includes passing the generic argument data structure to the invocation method.

[Action, page 11.] The Applicants agree. The Examiner argues, however, that Gardner demonstrates features such as "late bound interfaces and methods," "utilize[ing] packing and unpacking from a data structure," and type information, and that:

> [i]t would have been obvious to one of ordinary skill in the art at the time of the invention to implement the abstract class system of Williams with a description of dispatch late binding interface to implement as found in Gardner's teaching. This implementation would have been obvious because one of ordinary skill in the art would be motivated to provide object interface for as many environments as

possible in order to facilitate greater acceptance and use in the marketplace.... [Action, pages 11-12.]

Even if, for the sake of argument, Williams *could be* modified as suggested by the Examiner, this is not enough to make the Examiner's proposed modification obvious. [MPEP 2143.01; *see also* MPEP 2142.01 and 2145.X.C and D.] In fact, to the extent that Wiliams does discuss the use of interfaces, the Examiner's proposed modification *changes the principle of operation* of Williams and is thus improper. [*See In re Ratti*, 270 F.2d 810, MPEP § 2143.01.] In addition, Williams and Gardner teach away from the combination suggested by the Examiner.

Williams is directed toward generating and maintaining property sets. It is not directed toward compilation techniques or structures, and only briefly discusses compilation as a general background matter. Additionally, Williams contains only brief descriptions of the use of interfaces, as well as references to specific interfaces. Gardner, by contrast, describes the use of bridge software during runtime. There is no motivation in either Williams or Gardner to create a combination.

Furthermore, Williams describes interfaces whose function members are known at compile time, and for which persistent information is kept in order to facilitate runtime execution. [Williams, 3:1-22.] The Examiner's proposed modification would change this principle of operation of Williams. The Examiner proposes modifying Williams according to Gardner, so as to allow a client and server to communicate by using dispatchable interfaces for which interface information is *obtained at run time from a type library that is not created until run time*. [Gardner 8:7-14.] According to late binding as in Gardner the registry information of Williams would be unnecessary.

Williams use of persistent registry data to maintain interface information suggests, if nothing else, knowing interface function members before execution. Because of this, Williams teaches away from creation of a type library at run time whose purpose is to provide interface information, as this would be duplicative and inefficient. Thus, Williams teaches away from combination with Gardner, which uses these runtime techniques.

Finally, inasmuch as Gardner describes a type library implementation for a dispatch interface, it also leads away from the modification the Examiner has made. Creating a type library at run time leads away from modifying a compiler to generate dispatch interface implementations.

For at least these reasons, claim 23 should be allowable.

Claims 24, 25, and 30 (which depend from claim 23) should also be allowable, but the Applicants will not belabor the merits of the separate patentability of claims 24, 25, and 30.

CONCLUSION

Claims 1-30 should be allowed. Such action is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN, LLP

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 595-5300
Facsimile: (503) 595-5301

By _____
Kyle B. Rinehart
Registration No. 47,027